

Rael**Página Pessoal**[Go to home](#)[Anúncios Google](#) [PHP Download](#) [HTML Template](#) [PHP Template](#) [Download PH](#)

Tutorial de Templates em PHP

[< Voltar para post relacionado](#)

Uma das coisas que as pessoas mais me perguntam sobre PHP, é sobre o uso de Templates. Ou porque leram em algum livro ou fórum, ou seja porque eu comentei em aula.

Através do uso de templates, deixamos toda a estrutura visual (HTML ou XML, CSS, etc) separado da lógica de programação (código PHP), o que melhora e muito tanto a construção quanto a manutenção de sistemas web.

Existem vários mecanismo de template para PHP, e há um bom tempo no mercado. Mas qual indicar? Sem dúvida alguma, o [Smarty](#) é hoje o mais completo deles. Porém, o mais complexo, e de curva mais demorada. Ele é praticamente uma linguagem a parte do PHP.

Com base nisso, eu resolvi colocar criar um tutorial baseado em um mecanismo de template muito mais simples. Ou seja: este não é um tutorial sobre o [Smarty](#). É sobre Templates, mas baseado uma biblioteca que eu mesmo desenvolvi, e uso em meus projetos (mesmo nos maiores deles), na qual gastei bastante tempo desenvolvendo e melhorando, sempre com o foco principal na facilidade de uso.

Mas quase todos os conceitos vistos neste tutorial se aplicam a maioria das classes de Template existentes em PHP, portanto, se você entender as idéias explicadas aqui, poderá usar seu mecanismo de Templates favorito.

Para criar esta biblioteca (eu uso templates há muitos anos, quando só existia a PHPLib), eu estudei vários mecanismos de templates (PHPLib, Sigma, etc), portanto não estou "reinventando a roda", apenas "me apoiei no ombro de gigantes" para criar algo muito mais fácil de usar, que atendesse minhas necessidades e da empresa que eu trabalhava na época. De fato, atualmente muita gente usa esta biblioteca.

Eu inclusive traduzi os comentários dos métodos/funções públicos para facilitar. Você verá que será preciso entender apenas duas idéias básicas: variáveis e blocos.

Então, vamos lá.

Índice

[Histórico](#)
[Download, Documentação e Licença](#)
[Requisitos necessários](#)
[Instalação](#)
[Exemplo e explicação: Olá Mundo](#)
[Variáveis](#)
[Checando se Variáveis Existem](#)
[Blocos](#)
[Blocos aninhados](#)
[Blocos com HTML Select](#)
[Usando vários arquivos HTML](#)
[Guardando o conteúdo do template](#)
[Usando Objetos](#)
[Comentários](#)
[Criando XML, CSV e outros](#)
[Criando documentos do Office](#)
[Gerenciando erros](#)

[Variáveis dinâmicas](#)
[Mensagens de Erro](#)
[Precisão e Desempenho](#)
[Conclusão](#)

Histórico

26/02/2008 - Versão inicial do Tutorial
10/08/2008 - Adicionadas informações sobre html select
04/05/2009 - Renomeados: método parseBlock() para block(), getContent() para parse(), clearVar para clear()
05/05/2009 - Valor do parâmetro \$append do método block() alterado para true
07/05/2009 - Substituídas [Exceptions genéricas](#) pelas [Exceptions](#) da [SPL](#)
20/07/2009 - Adicionado suporte a comentários e pequena melhora no desempenho
06/04/2010 - Adicionado método exists() para checar existência de uma variável

Download, Documentação e Licença

Para baixar o código fonte, escolha entre a versão com comentários e mensagens de erro em português ou inglês:

- [Download Template \(português\)](#)
- [Download Template \(english\)](#)

Para ver a documentação dos métodos (API), clique no link do idioma desejado:

- [Ver API \(português\)](#)
- [Ver API \(english\)](#)

A licença desta biblioteca é regida pela licença [LGPL](#). Ou seja, você pode utilizá-la, como biblioteca, mesmo em projetos comerciais. Lembre-se apenas de ser uma pessoa legal e enviar de volta eventuais modificações, correções ou melhorias.

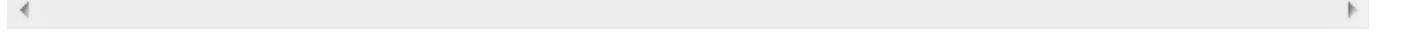
Requisitos Necessários

É preciso usar qualquer versão do PHP igual ou superior a 5. Versões 4 não são suportadas.

Instalação

Mais simples, impossível: apenas descompacte o arquivo, e o inclua no seu script PHP através da diretiva *require*:

```
<?php  
    require("Template.class.php");  
?>
```



Exemplo e explicação: Olá Mundo

O funcionamento básico do mecanismo de templates é baseado na seguinte idéia: você tem um arquivo

PHP (que usará a biblioteca), e este arquivo não conterá código HTML algum. O código HTML ficará separado, em um arquivo que conterá somente códigos HTML. Este arquivo HTML será lido pelo arquivo PHP. Aliás, tudo será processado no arquivo PHP.

Dito isso, vamos ao primeiro exemplo, o já manjado **Olá mundo**. Vamos criar 2 arquivos: o PHP responsável por toda a lógica, e o arquivo HTML com nosso layout.

Então, crie um arquivo HTML, chamado **hello.html** com o conteúdo abaixo:

```
<html>
<body>

    Olá Mundo, com templates PHP!

</body>
</html>
```

Agora, crie o arquivo PHP, **hello.php**:

```
<?php

    require("Template.class.php");
    $tpl = new Template("hello.html");
    $tpl->show();

?>
```

Agora basta executar em seu navegador, o script **hello.php**, e verificar que ele irá exibir o conteúdo lido de **hello.html**.

Se algo deu errado, consulte a seção sobre as [Mensagens de Erro](#).

Variáveis

Vamos agora a um conceito importante: *variáveis de template*. Como você pode imaginar, vamos querer alterar várias partes do arquivo HTML. Como fazer isso? Simples: no lado do HTML, você cria as chamadas *variáveis de template*. Veja o exemplo abaixo:

```
<html>
<body>

    Olá {FULANO}, com templates PHP!

</body>
</html>
```

Repare na variável FULANO, entre chaves. Ela vai ter seu valor atribuído no código PHP. Variáveis só podem conter em seu nome: letras, números e underscore (_). O uso de maiúsculas é apenas uma convenção, pra facilitar a identificação quando olhamos o código HTML. Mas, obrigatoriamente tem que estar entre chaves, sem nenhum espaço em branco.

Então, como ficaria o código PHP que atribui valor a ela? Vamos a ele:

```
<?php

    require("Template.class.php");
    $tpl = new Template("hello.html");
    $tpl->FULANO = "Rael";
```

```
$tpl->show();
```

?>

Execute então novamente o script, e você verá que o código final gerado no navegador será:

```
<html>
<body>
```

Olá Rael, com templates PHP!

```
</body>
</html>
```

Variáveis de template que não tiverem um valor atribuído, serão limpas do código final gerado. Outra coisa sobre variáveis: você pode repetir as variáveis de template (ou seja, usar a mesma variável em vários lugares). Mas, óbvio, todas mostrarão o mesmo valor.

Para ler o valor de uma variável, acesse do mesmo modo:

```
<?php
require("Template.class.php");
$tpl = new Template("hello.html");

// Atribuindo valor
$tpl->FULANO = "Rael";

// Imprimindo o valor da variável
die("Valor de FULANO: ".$tpl->FULANO);

$tpl->show();
```

?>

Repare que usando as variáveis de template, você pode continuar editando o arquivo HTML em seu editor favorito: as variáveis de template serão reconhecidas como um texto qualquer, e o arquivo HTML não ficará poluído de código PHP. O contrário também é verdade: seu arquivo PHP ficará limpo de código HTML.

Checando se Variáveis Existem

Caso você queira atribuir valor pra uma variável de template, mas não tem certeza se a variável existe, você pode usar o método [exists\(\)](#) para fazer essa checagem.

Como é de se esperar, ele retorna **true** caso a variável exista. Caso não, retorna **false**:

```
<?php
require("Template.class.php");
$tpl = new Template("layout.html");

// Checando existência da variável antes da atribuição
if($tpl->exists("FULANO")) $tpl->FULANO = "Rael";

$tpl->show();
```

?>

Blocos

Esse é o segundo e último conceito que você precisa saber sobre essa biblioteca de templates: os blocos. Imagine que você gostaria de listar o total de produtos cadastrados em um banco de dados. Se não houver nenhum produto, você irá exibir uma aviso de que nenhum produto foi encontrado.

Vamos utilizar então, dois blocos: um que mostra a quantidade total; e outro que avisa que não existem produtos cadastrados, caso realmente o banco esteja vazio. O código HTML para isso é:

```
<html>
<body>

    <p>Quantidade de produtos cadastrados no sistema:</p>

    <!-- BEGIN BLOCK_QUANTIDADE -->
    <div class="destaque">Existem {QUANTIDADE} produtos cadastrados.</div>
    <!-- END BLOCK_QUANTIDADE -->

    <!-- BEGIN BLOCK_VAZIO -->
    <div class="vazio">Não existe nenhum produto cadastrado.</div>
    <!-- END BLOCK_VAZIO -->

</body>
</html>
```

Repare que o início e final do bloco são identificados por um comentário HTML, com a palavra BEGIN (para identificar início) ou END (para identificar fim) e o nome do bloco em seguida. As palavras BEGIN e END sempre devem ser maiúsculas. O nome do bloco deve conter somente letras, números ou underscore.

E então, no lado PHP, vamos checar se os produtos existem. Caso sim, mostraremos o bloco BLOCK_QUANTIDADE. Caso não, vamos exibir o bloco BLOCK_VAZIO.

```
<?php

require("Template.class.php");
$tpl = new Template("hello.html");

// Vamos supor que esta quantidade veio do banco de dados
$quantidade = 5;

// Se existem produtos cadastrados, vamos exibir a quantidade
if($quantidade > 0){
    $tpl->QUANTIDADE = $quantidade;
    $tpl->block("BLOCK_QUANTIDADE");
}

// Caso não exista nenhum produto, exibimos a mensagem de vazio
else {
    $tpl->block("BLOCK_VAZIO");
}

$tpl->show();

?>
```

Como você pode reparar, blocos podem conter variáveis de template. E blocos só são exibidos se no código PHP pedirmos isso, através do método [block\(\)](#). Caso contrário, o bloco não é exibido no conteúdo final gerado.

Outro detalhe importante: ao contrário das variáveis de template, cada bloco deve ser único, ou seja, não podemos usar o mesmo nome para vários blocos.

Repare que mesmo com o uso de blocos, podemos continuar editando o arquivo HTML em qualquer editor HTML: os comentários que indicam início e fim de bloco não irão interferir em nada.

Agora vamos a outro exemplo usando blocos: imagine que você precisa mostrar os dados dos produtos que existem no seu cadastro. Vamos então, usando blocos, montar o HTML para isso:

```
<html>
<body>

    <p>Produtos cadastrados no sistema:</p>

    <table border=1>

        <tr><td>Nome</td><td>Quantidade</td></tr>

        <!-- BEGIN BLOCK_PRODUTO -->
        <tr>
            <td> {NOME} </td>
            <td> {QUANTIDADE} </td>
        </tr>
        <!-- END BLOCK_PRODUTO -->

    </table>

</body>
</html>
```

Repare que temos apenas uma linha de tabela HTML para os dados dos produtos, dentro de um bloco. Vamos então atribuir valor a estas variáveis, e ir duplicando o conteúdo do bloco conforme listamos os produtos:

```
<?php

require("Template.class.php");

$tpl = new Template("hello.html");

// Simulando produtos cadastrados no banco de dados
$produtos = array(
    array("nome" => "Sabão em Pó", "quantidade" => 15),
    array("nome" => "Escova de Dente", "quantidade" => 53),
    array("nome" => "Creme Dental", "quantidade" => 37)
);

// Listando os produtos
foreach($produtos as $p){
    $tpl->NOME = $p["nome"];
    $tpl->QUANTIDADE = $p["quantidade"];
    $tpl->block("BLOCK_PRODUTO");
}

$tpl->show();

?>
```

O comportamento padrão do método `block()` é manter o conteúdo anterior do bloco, somado (ou melhor, concatenado) ao novo conteúdo que acabamos de atribuir.

No exemplo acima, os dados dos produtos vieram do array `$produtos`. Caso estes dados estivessem armazenados em um banco de dados, então bastaríamos fazer como no exemplo abaixo:

```
<?php
```

```

require("Template.class.php");
$tpl = new Template("hello.html");
// ... Conectar ao banco, selecionar database, etc
// Produtos da database
$result = mysql_query("SELECT nome, quantidade FROM produtos");

// Listando os produtos
while($linha = mysql_fetch_array($result)){
    $tpl->NOME = $linha["nome"];
    $tpl->QUANTIDADE = $linha["quantidade"];
    $tpl->block("BLOCK_PRODUTO");
}
$tpl->show();
?>

```

Blocos aninhados

Vamos agora então juntar os 2 exemplos de uso de blocos que vimos: queremos mostrar os dados dos produtos em um bloco, mas caso não existam produtos cadastrados, exibiremos uma mensagem com um aviso. Vamos fazer isso agora usando blocos aninhados, ou seja, blocos dentro de outros blocos:

```

<html>
<body>

<p>Produtos cadastrados no sistema:</p>

<!-- BEGIN BLOCK_PRODUTOS -->
<table border=1>

<tr><td>Nome</td><td>Quantidade</td></tr>

<!-- BEGIN BLOCK_DADOS -->
<tr>
    <td>{NOME}</td>
    <td>{QUANTIDADE}</td>
</tr>
<!-- END BLOCK_DADOS -->

</table>
<!-- END BLOCK_PRODUTOS -->

<!-- BEGIN BLOCK_VAZIO -->
<div class=vazio>Nenhum registro encontrado.</div>
<!-- END BLOCK_VAZIO -->

</body>
</html>

```

E então, caso existam produtos, nós exibimos o bloco PRODUTOS. Caso contrário, exibimos o bloco VAZIO:

```

<?php
require("Template.class.php");

```

```
$tpl = new Template("hello.html");

// Produtos cadastrados
$produtos = array(
    array("nome" => "Sabão em Pó", "quantidade" => 15),
    array("nome" => "Escova de Dente", "quantidade" => 53),
    array("nome" => "Creme Dental", "quantidade" => 37)
);

// Listando os produtos
foreach($produtos as $p){
    $tpl->NOME = $p["nome"];
    $tpl->QUANTIDADE = $p["quantidade"];
    $tpl->block("BLOCK_DADOS");
}

// Se existem produtos, então mostramos o bloco com os dados de todos
if(isset($produtos) && is_array($produtos) && sizeof($produtos) > 0){
    $tpl->block("BLOCK_PRODUTOS");
}
// Senão, mostramos o bloco com o aviso de nenhum cadastrado
else {
    $tpl->block("BLOCK_VAZIO");
}

$tpl->show();

?>
```

Repare numa coisa muito importantes nesse exemplo: para que um bloco seja mostrado, é preciso que o bloco externo a ele também o seja. Senão, o bloco interno não é exibido. Podemos também criar quantos blocos aninhados acharmos necessário, desde que sigamos esse princípio: se quisermos que um deles seja mostrado, tanto ele quanto o bloco externo (se existir) devem ser exibidos.

Blocos com HTML Select

Uma das dúvidas mais comuns é: como usar a classe Template com o elemento Select do HTML? Ou melhor: como fazer um elemento Option ficar selecionado, usando Template?

Vamos então montar nossa página HTML com o elemento Select e os devidos Options, representando cidades de uma lista:

```
<html>
<body>

<select name="cidades">

    <!-- BEGIN BLOCK_OPTION -->
    <option value="{VALUE}" {SELECTED}>{TEXT}</option>
    <!-- END BLOCK_OPTION -->

</select>

</body>
</html>
```

Agora vamos ao respectivo arquivo PHP:

```
<?php
    require_once("Template.class.php");
```

```
$tpl = new Template("index.html");

// Array de cidades
$cidades = array(0 => "Cidade 0", 1 => "Cidade 1", 2 => "Cidade 2");

// Valor selecionado
$atual = 1;

foreach($cidades as $value => $text){

    $tpl->VALUE = $value;
    $tpl->TEXT = $text;

    // Vendo se a opção atual deve ter o atributo "selected"
    if($atual == $value) $tpl->SELECTED = "selected";

    // Caso esta não seja a opção atual, limpamos o valor da variável SELECTED
    else $tpl->clear("SELECTED");

    $tpl->block("BLOCK_OPTION");
}

$tpl->show();

?>
```

Como resultado, o navegador exibirá o seguinte código:

```
<html>
<body>

<select name="cidades">

    <option value="0" >Cidade 0</option>
    <option value="1" selected>Cidade 1</option>
    <option value="2" >Cidade 2</option>

</select>

</body>
</html>
```

Reparou que no arquivo PHP chamamos o método `clear`? Se não chamarmos este método (que limpa o valor de uma variável), todas as opções (Options) ficariam com a propriedade "selected" (obviamente, efeito não desejado):

```
<html>
<body>

<select name="cidades">

    <option value="0" selected>Cidade 0</option>
    <option value="1" selected>Cidade 1</option>
    <option value="2" selected>Cidade 2</option>

</select>

</body>
</html>
```

Usando vários arquivos HTML

Um uso bastante comum de templates é usarmos um arquivo HTML que contenha a estrutura básica do nosso site: cabeçalho, rodapé, menus, etc. E outro arquivo com o conteúdo da página que desejamos mostrar, ou seja, o "miolo". Dessa forma, não precisamos repetir em todos os arquivos HTML os elementos comuns (cabeçalho, rodapé, etc), e as páginas HTML que terão o conteúdo (o "miolo") ficarão mais limpas, menores e mais fáceis de serem mantidas.

Como fazer isso com templates? Em primeiro lugar, vamos criar nosso arquivo "base" HTML, o arquivo **base.html**:

```
<html>
<head>
<title>Título da Página</title>
</head>
<body>

<div>{FULANO}, seja bem vindo!</div>

<div>{CONTEUDO}</div>

<div>Deseja maiores informações? Clique <a href="info.php">aqui</a> para saber</div>

</body>
</html>
```

Agora, vamos criar o arquivo que contém o "miolo" de nossa página HTML, o arquivo **miolo.html**:

```
<p>Produtos cadastrados no sistema:</p>

<!-- BEGIN BLOCK_PRODUTOS -->
<table border=1>

<tr><td>Nome</td><td>Quantidade</td></tr>

<!-- BEGIN BLOCK_DADOS -->
<tr>
    <td> {NOME} </td>
    <td> {QUANTIDADE} </td>
</tr>

<!-- END BLOCK_DADOS -->

</table>
<!-- END BLOCK_PRODUTOS -->

<!-- BEGIN BLOCK_VAZIO -->
<div class=vazio>Nenhum registro encontrado.</div>
<!-- END BLOCK_VAZIO -->
```

No arquivo PHP então, usamos o método `addFile()`, onde informamos duas coisas: em qual variável do template o conteúdo do novo arquivo será jogado, e qual o caminho desse arquivo. Depois disso, basta usar as variáveis e blocos normalmente, independente de qual arquivo HTML eles estejam:

```
<?php
    require("Template.class.php");
    $tpl = new Template("base.html");
```

```
// Adicionando mais um arquivo HTML
$tpl->addFile("CONTEUDO", "miolo.html");

$tpl->FULANO = "Rael!";

// Produtos cadastrados
$produtos = array(
    array("nome" => "Sabão em Pó", "quantidade" => 15),
    array("nome" => "Escova de Dente", "quantidade" => 53),
    array("nome" => "Creme Dental", "quantidade" => 37)
);

// Listando os produtos
foreach($produtos as $p){
    $tpl->NOME = $p["nome"];
    $tpl->QUANTIDADE = $p["quantidade"];
    $tpl->block("BLOCK_DADOS");
}

// Se existem produtos, então mostramos o bloco com os dados de todos
if(isset($produtos) && is_array($produtos) && sizeof($produtos) > 0){
    $tpl->block("BLOCK_PRODUTOS");
}
// Senão, mostramos o bloco com o aviso de nenhum cadastrado
else {
    $tpl->block("BLOCK_VAZIO");
}

$tpl->show();

?>
```

Guardando o conteúdo do template

Até agora exibimos o conteúdo gerado pelo template na tela, através do método [show\(\)](#). Mas, e quisermos fazer outro uso para esse conteúdo, como salvá-lo em arquivo ou outra coisa do tipo? Basta usarmos o método [parse\(\)](#), que gera o conteúdo final e o retorna:

```
<?php

require("Template.class.php");

$tpl = new Template("base.html");
$tpl->addFile("CONTEUDO", "miolo.html");

// Variáveis, blocos, etc
$tpl->FULANO = "Rael!";

// Pega o conteúdo final do template
$conteudo = $tpl->parse();
// Salva em um arquivo
file_put_contents("arquivo.txt", $conteudo);

?>
```

Usando Objetos

A classe Template suporta a atribuição de objetos para as variáveis de template em sua versão atual: versão 1.5 (verifique no código fonte da classe sua versão).

Isso faz com que o código nos arquivos PHP fique bastante reduzido (claro, desde que você use objetos), e devido a isso, há uma melhora (quase imperceptível) em desempenho.

Para que os exemplos fiquem mais claros, vamos trabalhar com uma suposta página que exibe detalhes de um produto. Os produtos possuem como atributos: **nome**, **id** e **valor**.

Temos uma classe PHP chamada **Produto**, que representa a tabela **produtos** do banco de dados, com o seguinte código (bem rudimentar e sem métodos para salvar/atualizar os dados):

```
<?php

class Produto {

    protected $id = 0;
    protected $name;
    protected $price = 0.0;

    public function __construct($id = 0){
        if(0!=$id) $this->loadById($id);
    }

    public function loadById($id){
        $db = new PDO('mysql:host=localhost;dbname=database', 'usuario', 'senha');
        $result = $db->query("SELECT * FROM produtos WHERE id = ".intval($id));
        if($row = $result->fetch()){
            $this->setId($row['id']);
            $this->setName($row['nome']);
            $this->setPrice($row['preco']);
            return true;
        }
        return false;
    }

    public function getId(){
        return $this->id;
    }

    public function setId($id){
        $this->id = intval($id);
    }

    public function getName(){
        return $this->name;
    }

    public function setName($name){
        $this->name = $name;
    }

    public function getPrice(){
        return $this->price;
    }

    public function setPrice($price){
        $this->price = floatval($price);
    }
}

?>
```

Usando a classe acima, esta é a forma como estamos fazendo até agora para montar uma página PHP que exibe os dados do produto:

```
<?php

require("Template.class.php");
$tpl = new Template("produtos.html");
```

```
// Id do produto vem por GET
$p = new Produto($_GET['id']);

$tpl->NAME = $p->getName();
$tpl->ID = $p->getId();
$tpl->PRICE = $p->getPrice();

$tpl->show();

?>
```

Agora o respectivo arquivo HTML:

```
Nome: {NAME} <br/>
Id: {ID} <br/>
Preço: {PRICE} <br/>
```

Vamos então modificar o arquivo PHP para usar o suporte a objetos de Template:

```
<?php

require("Template.class.php");
$tpl = new Template("produtos.html");

// Id do produto vem por GET
$tpl->P = new Produto($_GET['id']);

$tpl->show();

?>
```

O arquivo HTML também deve ser modificado pra exibir as propriedades de Produto:

```
Nome: {P->NAME} <br/>
Id: {P->ID} <br/>
Preço: {P->PRICE} <br/>
```

A instrução **P->NAME** chamará o método **\$p->getName()**, caso ele exista. Se não existir esse método na classe, um erro será disparado.

Isso vale para qualquer atributo que tentarmos chamar no HTML: será traduzido para **\$meuObjeto->getAtributo()**.

Se o nome do método PHP for composto, como por exemplo **\$p->getExpirationDate()**, basta usar underscore (_) no HTML como separador dos nomes: no caso do exemplo, ficaria **P->EXPIRATION_DATE**.

A classe Template também entende a existência do método [toString\(\)](#).

Comentários

A partir da versão 1.8 (veja no código fonte da classe o número da versão), a classe Template, a exemplo das linguagens de programação, suporta comentários de Template no HTML. Comentários são úteis para várias coisas, entre elas, identificar o autor do HTML, versão, incluir licenças, etc.

Diferentemente dos comentários HTML, que são exibidos no código fonte da página, os comentários da

classe Template são extraídos do HTML final. Na verdade os comentários de Template são extraídos antes mesmo de qualquer processamento, e tudo que estiver entre os comentários será ignorado.

Os comentários ficam entre as tags `<!-- e -->`. Repare que usamos 3 tracinhos, ao invés de 2 (que identificam comentários HTML). A razão é simples: permitir diferenciarmos entre um e outro, e permitir que os editores continuem reconhecendo o conteúdo entre `<!-- e -->` como comentários.

Veja o exemplo abaixo:

```
<!--
Listagem de produtos.

@author Rael
@version 1.0
-->

<p>Produtos cadastrados no sistema:</p>

<table border=1>

<tr><td>Nome</td><td>Quantidade</td></tr>

<!-- BEGIN BLOCK_DADOS -->
<tr>
    <td> {NOME} </td>
    <td> {QUANTIDADE} </td>
</tr>

<!-- END BLOCK_DADOS -->

</table>
<!-- END BLOCK_PRODUTOS -->

<!-- BEGIN BLOCK_VAZIO -->
<div class=vazio>Nenhum registro encontrado.</div>
<!-- END BLOCK_VAZIO -->
```

Criando XML, CSV e outros

O uso mais comum de templates é com arquivos HTML. Mas como essa biblioteca é direcionada ao uso de qualquer tipo de arquivo de texto, podemos usá-la com vários outros formatos de arquivo, como XMLs e arquivos CSVs.

Como fazer isso? Mais simples impossível: não muda nada, basta apenas ao invés de indicar um arquivo HTML para o Template, indicar qualquer outro arquivo de texto. E usar variáveis e blocos nele conforme já vimos, exibindo o conteúdo na tela, ou salvando em arquivos.

Criando arquivos do Office

Se você precisa elaborar um relatório que deve ser exibido em formato do Word (.doc) ou do Excel (.xls), também podemos usar a classe Template para isso.

Em primeiro lugar, crie normalmente no Office seu relatório. Após terminar, escolha a opção "Salvar como", e selecione o formato HTML. Feito isso, abra este arquivo HTML em seu editor PHP (não se assuste, é bastante poluído e cheio de tags estranhas) e use-o conforme visto até agora: crie variáveis, declare blocos, nada de diferente.

Se você for salvar o conteúdo em um arquivo, coloque neste arquivo a extensão .doc" (ou .xls no caso

de uma planilha). O Office abrirá normalmente este arquivo, convertendo-o automaticamente de HTML para o formato desejado na primeira vez em que for aberto.

Se você for exibir o conteúdo no navegador ao invés de salvá-lo num arquivo, você precisa modificar o header para avisar o navegador que se trata de um documento do Office, forçando o navegador a interpretá-lo como tal (o Firefox irá fazer o download do arquivo, o IE irá abrir o Microsoft Office como um plugin e exibir o arquivo dentro do navegador mesmo). Faça isso com a instrução **header()** do PHP:

```
<?php

require("Template.class.php");

// Forçando o cabeçalho para o formato escolhido do Office
header('Content-type: application/msword');
header('Content-Disposition: attachment; filename="Relatorio.doc"');
header("Pragma: no-cache");
header("Expires: 0");

// Arquivo relatorio.html, gerado no Word
$tpl = new Template("relatorio.html");

// Variáveis, blocos, etc
$tpl->FULANO = "Rael";

$tpl->show();

?>
```

Gerenciando erros

Quando um erro acontece, você deve ter reparado que a mensagem de erro gerada pela classe Template é um pouco diferente do usual em PHP: ao invés de ser apenas um **die()** com a mensagem de erro, é gerada uma exceção (Exception). Por que isso?

Com as exceptions, temos duas vantagens: a primeira é ver todo o *stack* do erro, ou seja, ver desde o lugar em que foi originado o erro, passando por todos os arquivos em que ele apareceu. Isso facilita muito o trabalho de *debug* e correção. Para ver o código de erro e a *stack* corretamente, peça para o navegador exibir o código-fonte da página, pois somente neste modo as quebras de linhas são visualizadas corretamente.

A segunda vantagem é poder gerenciar o erro, se desejarmos, e fazermos com que a execução de nosso script não seja interrompida, através do uso de [try/catch](#):

```
<?php

require("Template.class.php");
$tpl = new Template("index.html");

// Tentando acessar variável que não existe
try {

    $tpl->FOO = "bar";

    // Capturando erro e evitando que o script seja interrompido
} catch (Exception $e){

    echo "FOO não existe!";

}

$tpl->show();

?>
```

Variáveis Dinâmicas

Imagine um caso onde você tem várias variáveis de template em seu arquivo HTML, e tem um valor que precisa atribuir a uma delas. Mas o problema é: a variável que precisa receber o valor é definida apenas durante a execução do script. Ou seja, é uma variável dinâmica, ou como alguns chamam, uma "variável variável". Nosso arquivo HTML então seria:

```
<html>
<body>
    Olá {NOME_FULANO}!
</body>
</html>
```

Repare que no arquivo HTML não há nada de diferente. No arquivo PHP então, basta usar chaves:

```
<?php
require("Template.class.php");
$tpl = new Template("base.html");
// Nome da variável
$varname = "fulano";
// Variável definida dinamicamente
$tpl->{"NOME_".strtoupper($varname)} = "Rael";
$tpl->show();
?>
```

Mensagens de Erro

Abaixo estão os significados para as mensagens de erro exibidas pela classe Template.

- Parse error: syntax error, unexpected T_STRING, expecting T_OLD_FUNCTION or T_FUNCTION or T_VAR or '}'**: provavelmente você está usando PHP 4 (veja os [requisitos necessários](#) para usar esta biblioteca).
- addFile: var <varname> não existe**: você está usando o método [addFile\(\)](#) para adicionar um arquivo HTML (ou equivalente), mas a variável de template na qual você quer jogar o conteúdo, não existe.
- var <varname> não existe**: você está tentando atribuir valor a uma variável que não existe. Certifique-se de que o nome da variável de template está correto, e que você está utilizando como nome desta variável somente letras, números e underscore, entre chaves.
- arquivo <filename> não existe**: você está informando o caminho para um arquivo HTML (ou equivalente) que não existe, ou cuja permissão de leitura é negada.
- arquivo <filename> está vazio**: o arquivo HTML (ou equivalente) que você está passando como parâmetro está vazio. Se está vazio, ou você está informando um arquivo errado, ou esqueceu de colocar conteúdo nele.
- bloco duplicado: <blockname>**: o nome que você está tentando atribuir ao bloco já foi dado para outro bloco. Lembre-se que o nome dos blocos deve ser único. Se você estiver usando mais de um arquivo HTML (ou equivalente), o bloco com o mesmo nome que o seu pode estar em um

dos outros arquivo.

7. **bloco <blockname> está mal formado:** o bloco que você declarou está com defeitos. Talvez você tenha usado a tag BEGIN BLOCK com um nome, e tenha terminado (a tag END BLOCK) com outro. Ou então, esqueceu da tag END BLOCK.
 8. **bloco <blockname> não existe:** você está informando ao método [block\(\)](#) o nome de um bloco que não existe. Certifique-se de que o nome do bloco está correto, e que você está utilizando como nome deste bloco somente letras, números e underscore.
 9. **não existe método na classe <classname> para acessar <objeto>-><propriedade>:** não existe método para acessar a propriedade que você está chamando. Se você chamar no HTML por **OBJETO->NOME**, a classe deste objeto precisa ter um método chamado [getNome\(\)](#) ou [isNome\(\)](#). Veja maiores detalhes na seção "[Usando Objetos](#)".
-

Precisão e Desempenho

Uma dúvida comum ao uso de templates é: o quanto isso afeta o desempenho? A resposta é: menos do que você imagina. Deixe-me explicar o motivo.

Esta biblioteca é muito mais rápida que as bibliotecas de template antigas, como a finada PHPLib, e outras similares. O motivo é que o uso de expressões regulares dentro dela é evitado, não sendo usado para fazer todo o funcionamento, como as bibliotecas antigas faziam.

Ainda assim, esta biblioteca não possui um mecanismo de cache, como por exemplo, o template [Smarty](#) possui. Por duas razões. A primeira é a simplicidade de uso: não é preciso criar nem configurar diretórios, nem permissões Unix, etc. A segunda é que o desempenho é bom o suficiente para manter esta simplicidade. De qualquer forma, sinta-se à vontade para criar um mecanismo de cache.

Eu uso esta biblioteca durante anos, e fui aperfeiçoando seu desempenho. Ela já foi usada em sites de grande tráfego, e nunca foi o gargalo. Se você usa esta biblioteca e fizer uma medição de desempenho de seu sistema, verá que em 98% dos casos, o gargalo é o banco de dados, e não o uso do processador. Sempre são consultas mal construídas, ou consultas feitas dentro de laços, este tipo de coisa.

Um efeito colateral do bom desempenho desta biblioteca: se você pedir para seu navegador exibir o código fonte de uma página gerada pela classe Template, irá ver que o código final não remove algumas tabulações (tab ou \t) que existiam no começo de cada bloco do template, dando a aparência de que o código final está um pouco mais bagunçado do que deveria. A razão disto é que se essas tabulações fossem removidas por padrão, esta remoção traria uma queda na performance da classe Template.

Como uma tabulação no código fonte não traz efeito algum para o conteúdo HTML final, o comportamento padrão da classe Template é ignorar estas tabulações de início de bloco, deixando elas no código final. O único caso em que isso pode ser um problema é quando você precisa de uma reprodução fiel dos seus arquivos HTML, como no uso das tags `<pre>` e `<code>`. Já prevendo isso, existe um segundo parâmetro (opcional) usado na declaração do objeto Template: o parâmetro **\$accurate**. Se você usar ele com o valor **true**, então seu código final HTML será uma reprodução fiel dos arquivos HTML usados no Template (com a devida penalidade em performance):

```

<?php
require("Template.class.php");

// Parâmetro $accurate com valor TRUE
$tpl = new Template("base.html", true);

// ...
$tpl->show();

?>

```

Conclusão

O uso de mecanismos de Template é um grande avanço no desenvolvimento de aplicações web, pois nos permite manter a estrutura visual de nosso aplicativo separado da programação PHP.

Eu tentei incluir neste tutorial todos os tópicos que cobrem o uso de templates. Se você tiver problemas, procure primeiro em todos os tópicos aqui. Lembre-se que este trabalho é voluntário, e que eu gastei muito tempo escrevendo este tutorial, além do tempo gasto nesta biblioteca. Portanto, antes de me enviar um email com algum problema, tente resolvê-lo sozinho: grande parte do aprendizado está nisso. Se você não conseguir, vá em frente, e fale comigo.

Use os [comentários](#) para dizer se este tutorial lhe foi útil ou não, sugerir melhorias ou correções. Se você encontrou uma maneira de fazer algo melhor, seja bonzinho e me avise, eu publicarei aqui com os devidos créditos.

[Anúncios Google](#) [PHP Download](#) [HTML Template](#) [PHP Template](#) [Download PH](#)

[Link](#) - [Comentários\(165\)](#) - [Comentar](#)



Developed by myself :)